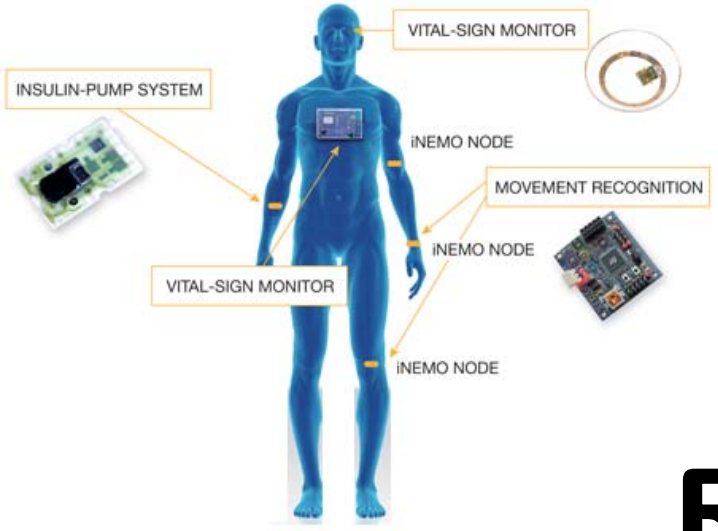
Serial Ports



Real Time Embedded Systems

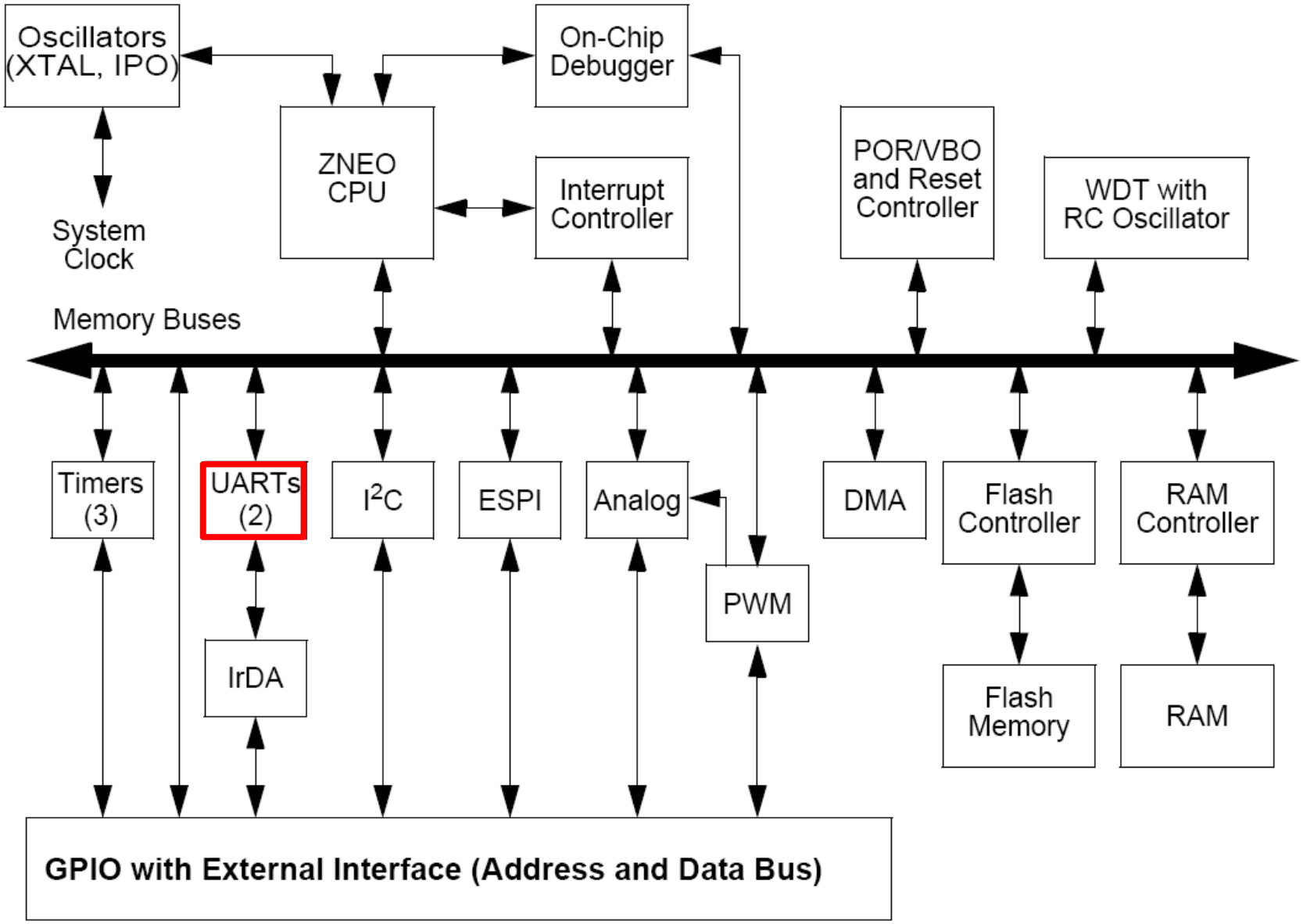
www.atomicrhubarb.com/embedded

Lecture 1 – January 17, 2012

Topic

Section Topic

- Where in the books
  - Catsoulis chapter/page
  - Simon chapter/page
  - Zilog UM197 (ZNEO Z16F Series Flash Microcontroller Contest Kit User Manual)
  - Zilog UM171 (ZiLOG Developer Studio II—ZNEO User Manual)
  - Zilog PS220 (ZNEO Z16F Series Product Specification)
  - Zilog UM188 (ZNEO CPU Core User Manual)
  - Assorted datasheets


Oscillators (XTAL, IPO)
On-Chip Debugger
ZNEO CPU
Interrupt Controller
POR/VBO and Reset Controller
WDT with RC Oscillator
System Clock
Memory Buses
Timers (3)
UARTs (2)
I2C
ESPI
Analog
DMA
Flash Controller
RAM Controller
PWM
IrDA
Flash Memory
RAM
GPIO with External Interface (Address and Data Bus)

Serial Interfaces/Buses

- Asynchronous
  - Serial Ports
    - RS-232, RS-422
  - UARTs
  - 1-wire
- Synchronous
  - I^2C – Inter Integrated Circuit
  - SPI – Serial Peripheral Interface
  - 3-Wire, Microwire

What do we mean by “Synchronous” and “Asynchronous”?

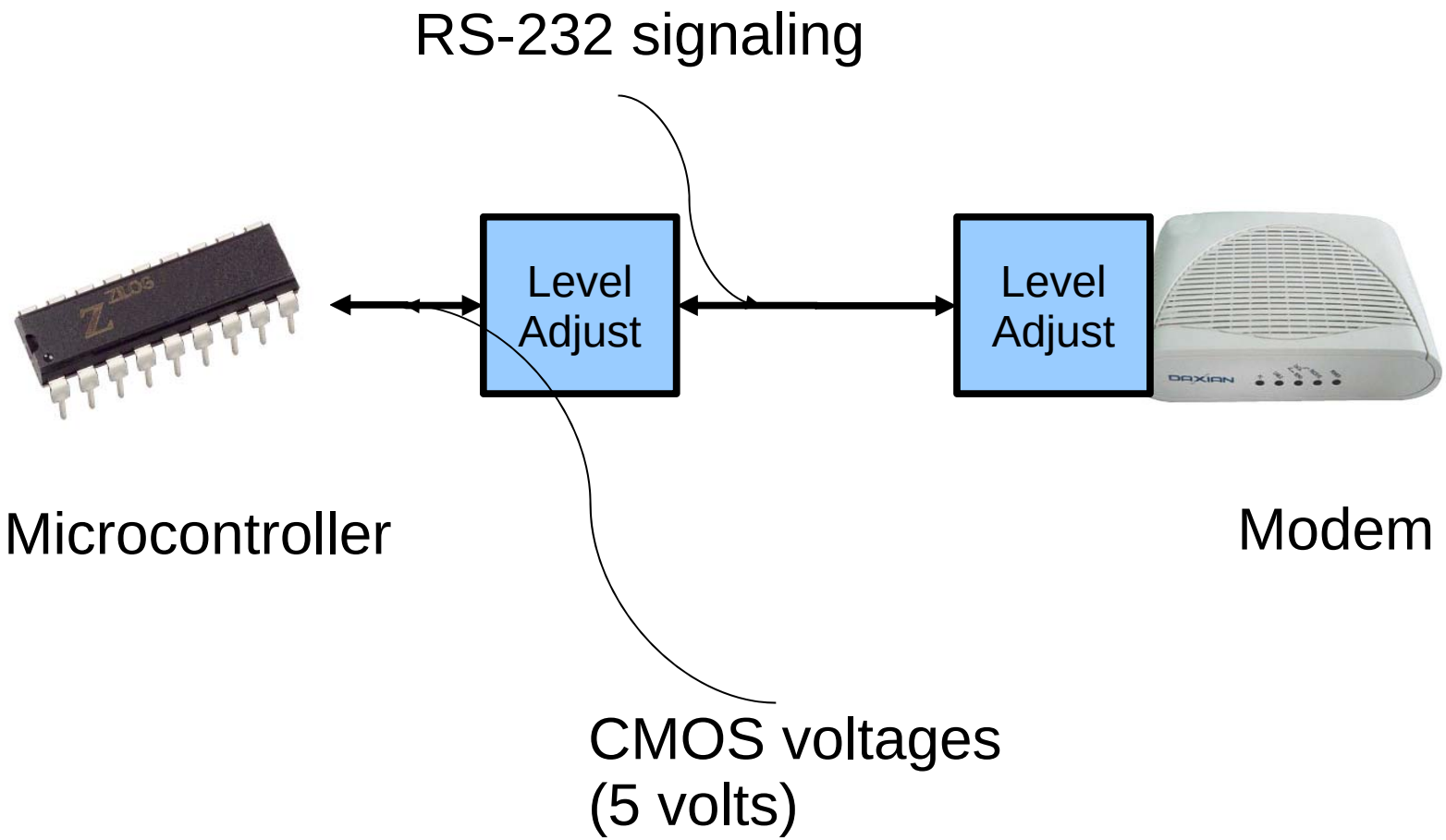
Serial Ports

Computer

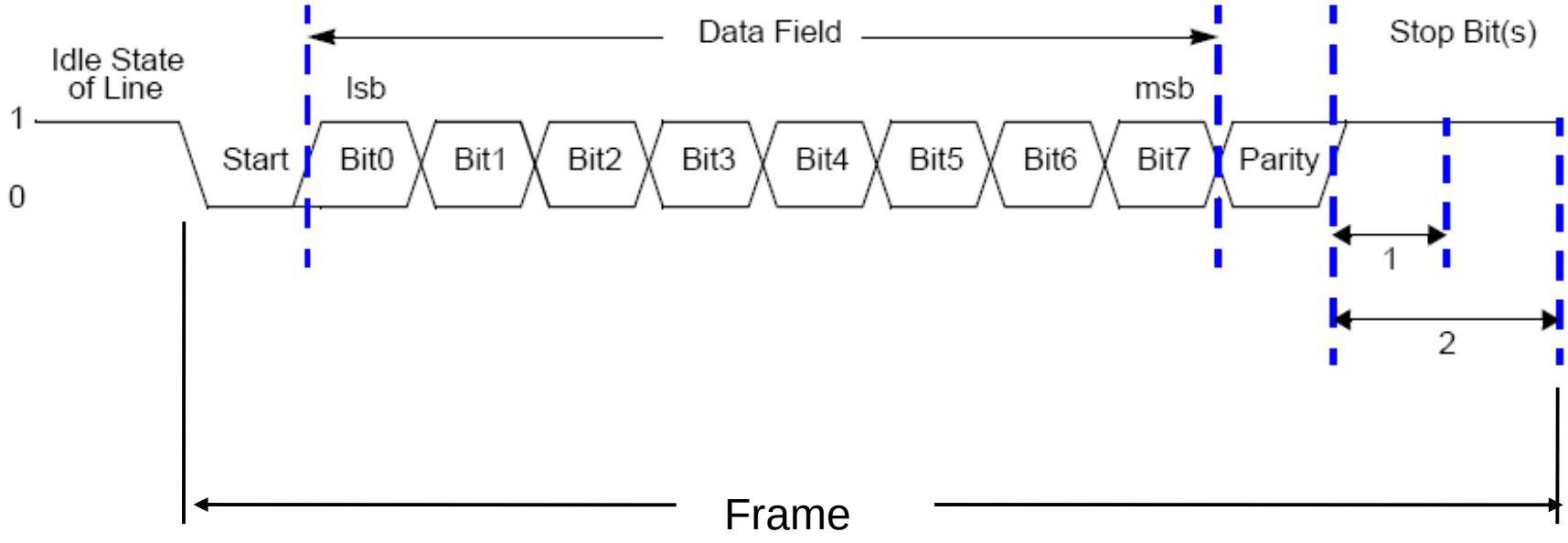
Modem

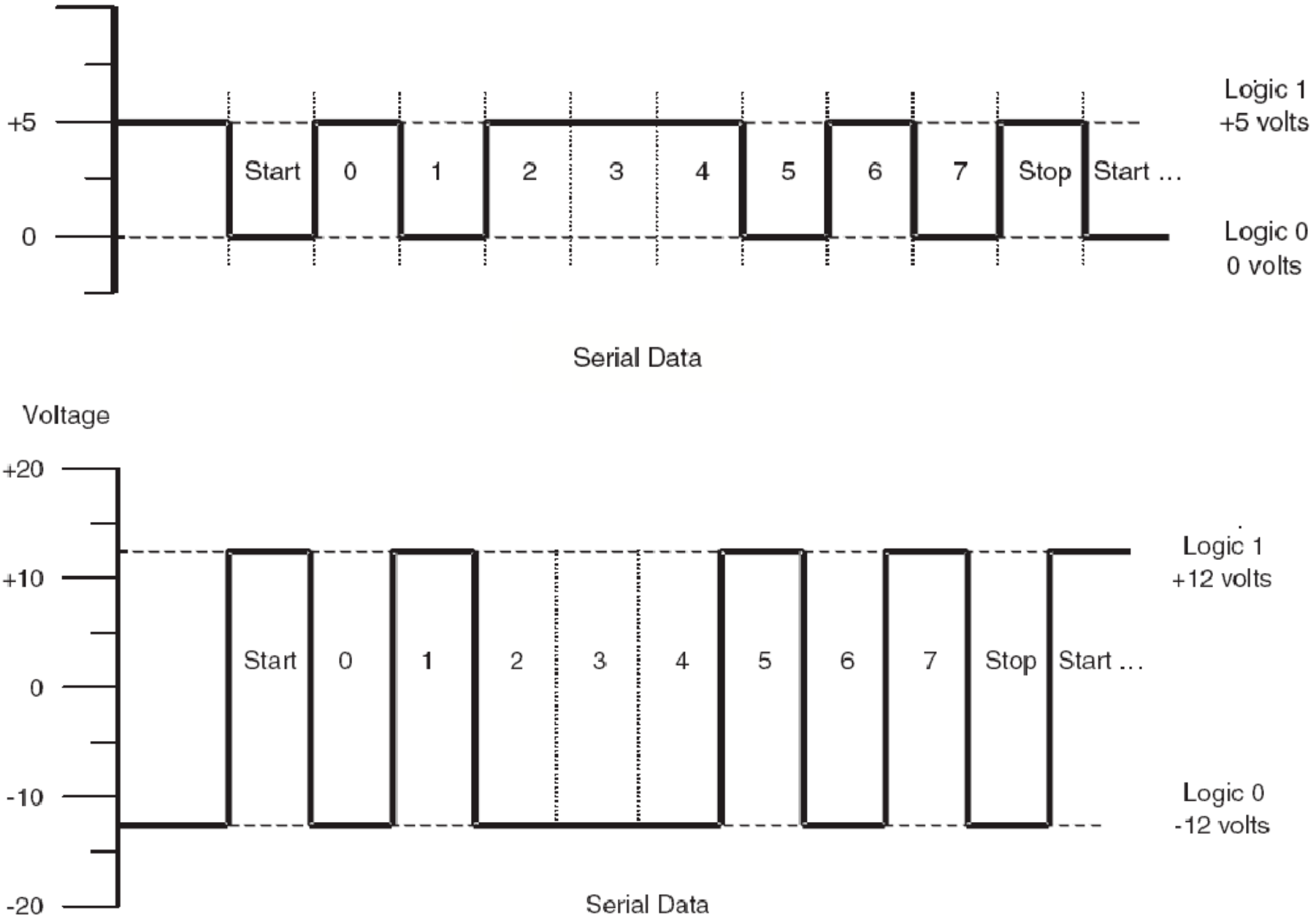
RS-232 signals
(inverted logic, +/- 12 volts)

Serial Ports



Serial data format




+5
0
Start
0
1
2
3
4
5
6
7
Stop
Start ...
Logic 1
+5 volts
Logic 0
0 volts
Serial Data
Voltage
+20
+10
0
-10
-20
Start
0
1
2
3
4
5
6
7
Stop
Start ...
Logic 1
+12 volts
Logic 0
-12 volts
Serial Data

Device

- RS-232 intended for connecting Terminals to Modems.
- DTE – Data Terminal Equipment
- DCE – Data Communication Equipment

Serial Connections

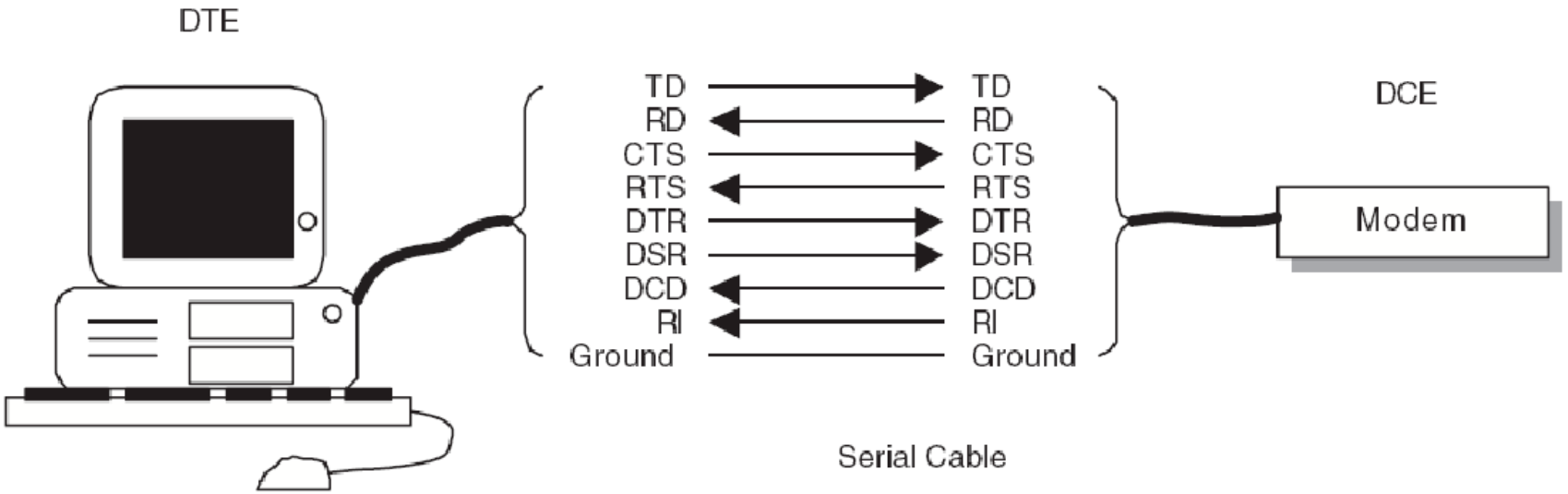


Figure 9-5: Serial data lines

Serial Signal Names

- TXD – Transmit data. This is the serial encoded data sent from a DTE to a DCE device.
- RXD – Receive data. This is the serial encoded data received by a DTE from a DCE device.
- DCD – Data Carrier Detect. This is set true by the DCE when it detects the data carrier signal on the telephone line (for modems; it may have other meanings for other devices or it may be ignored). Active high.
- DTR – Data Terminal Ready. This should be set true by a DTE whenever it is powered on and ready to communicate. It can be read by the DCE to determine that the DTE is ready. Active high.
- DSR – Data Set Ready. This should be set true by DCE whenever it is powered on and ready. It can be read by the DTE to determine if the DCE is ready. Active high.
- RTS – Request To Send. This is set true by the DTE when it wishes to transmit data. Active high.
- CTS – Clear To Send. This is set true by DCE to notify the DTE that it can transmit data. Active high.

DTE-DCE

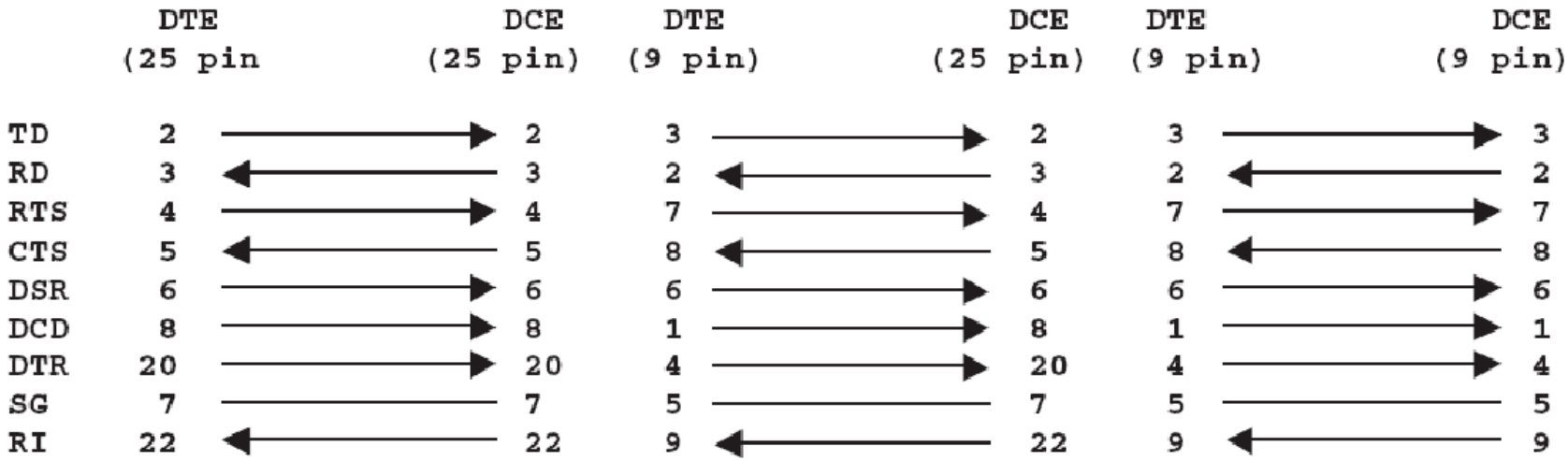


Figure 9-8: DTE–DCE cables (straight-through wired)

DTE-DTE

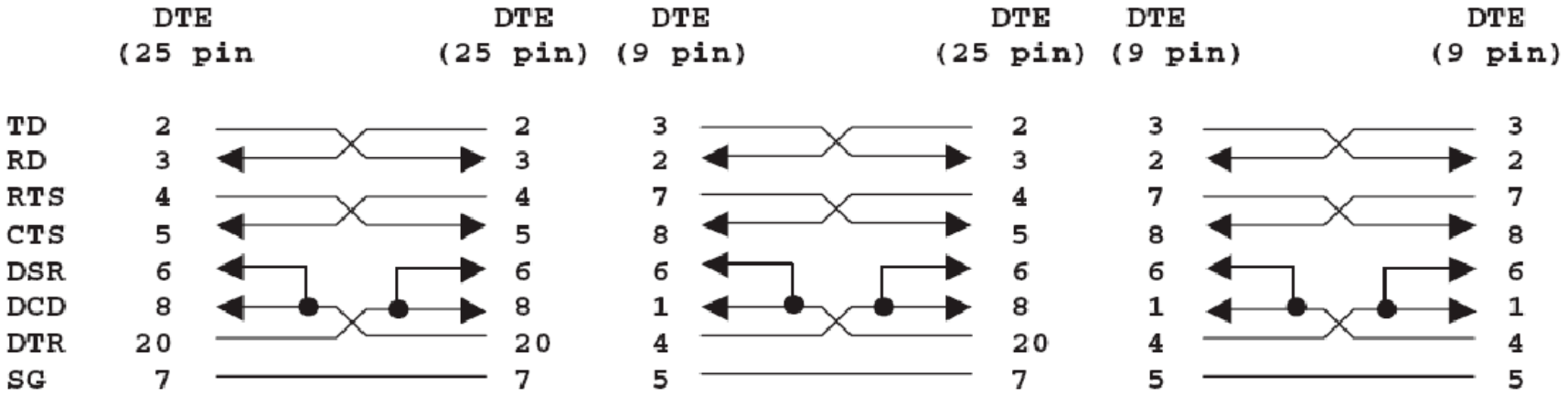
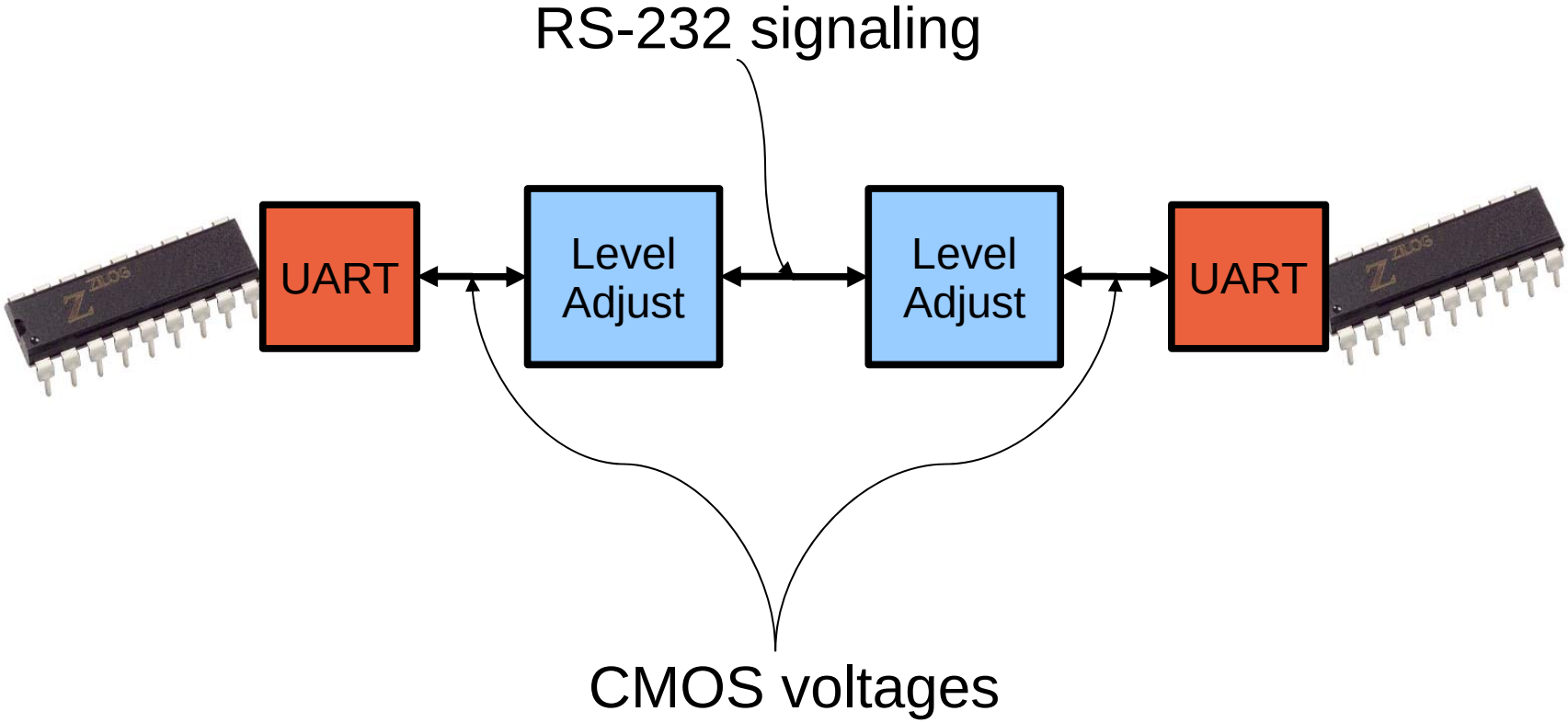


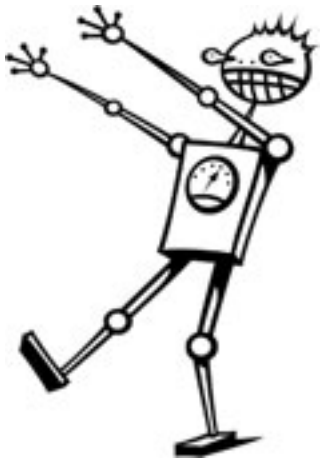
Figure 9-9: DTE-to-DTE cables (null modem cables or adapters)

Revised View



UART

- Universal Asynchronous Receiver-Transmitter, the component that handles asynchronous serial communication.
- Asynchronous - Not synchronized. Data is transmitted without an accompanying clock signal.



UART

- The UART takes bytes of data and transmits the individual bits in a sequential fashion.
- Provides additional circuits for control signals that can be used to indicate the state of the transmission media (status), and to regulate the flow of data (handshake).

UART

- Formatting includes adding of START and STOP bits, managing Parity, managing Baud Rate.
- UART does **not** do the level shifting.

UART Summary

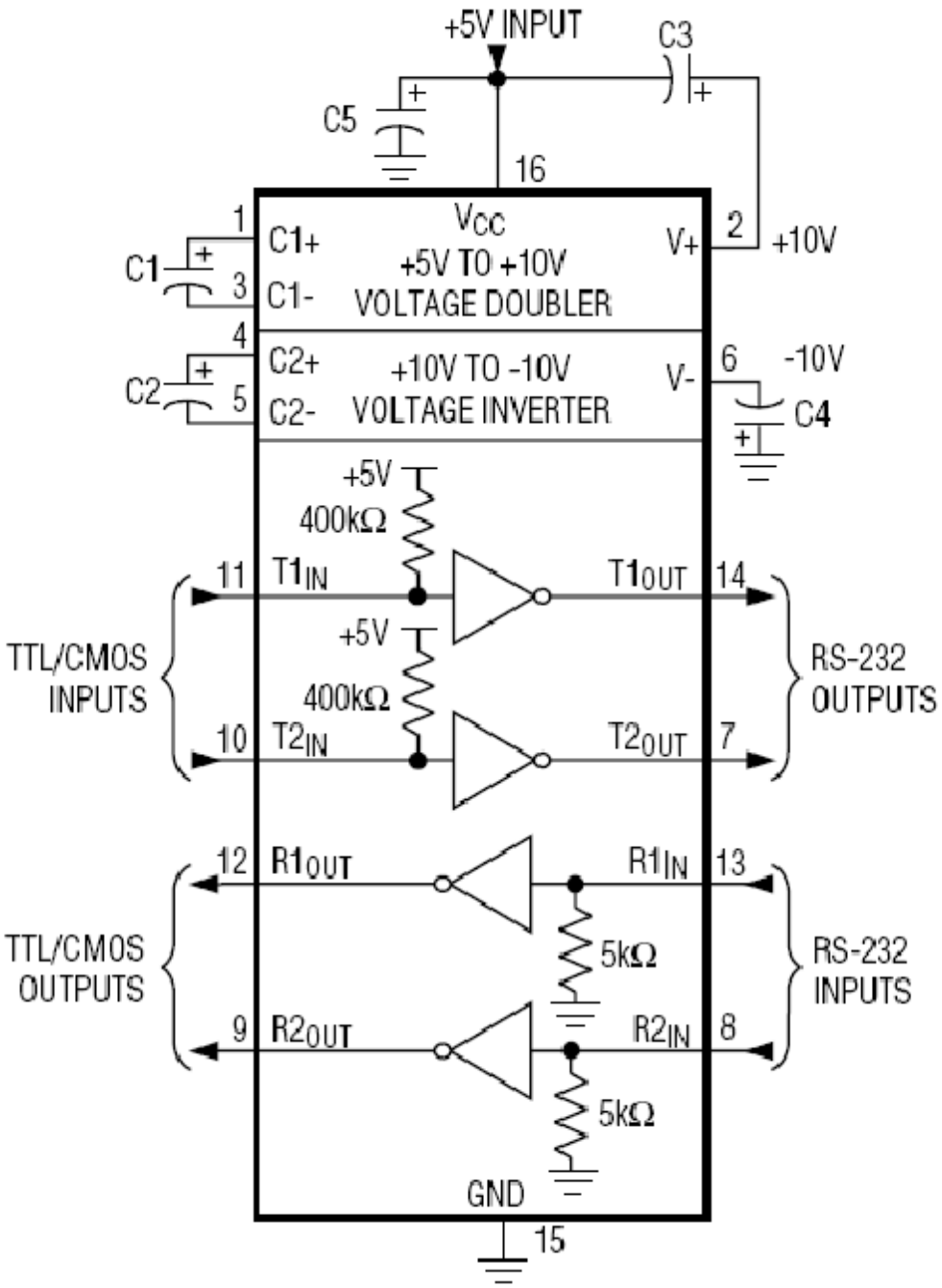
- Data is formatted by UART
- UART manages data Transmit and Receive
- UART manages handshake and status
- Level Shifter is required to get to/from RS-232 levels from/to CMOS logic levels.

Level Shifters

- RS232C, requires negative logic, i.e., logic '1' is -3V to -12V and logic '0' is +3V to +12V.
- To convert a CMOS logic levels (TxD and RxD) from the microcontroller we need a converter chip (serial line driver).
- A MAX232 is one chip that is common. There are many.

MAX232

- Serial
  Line Drive

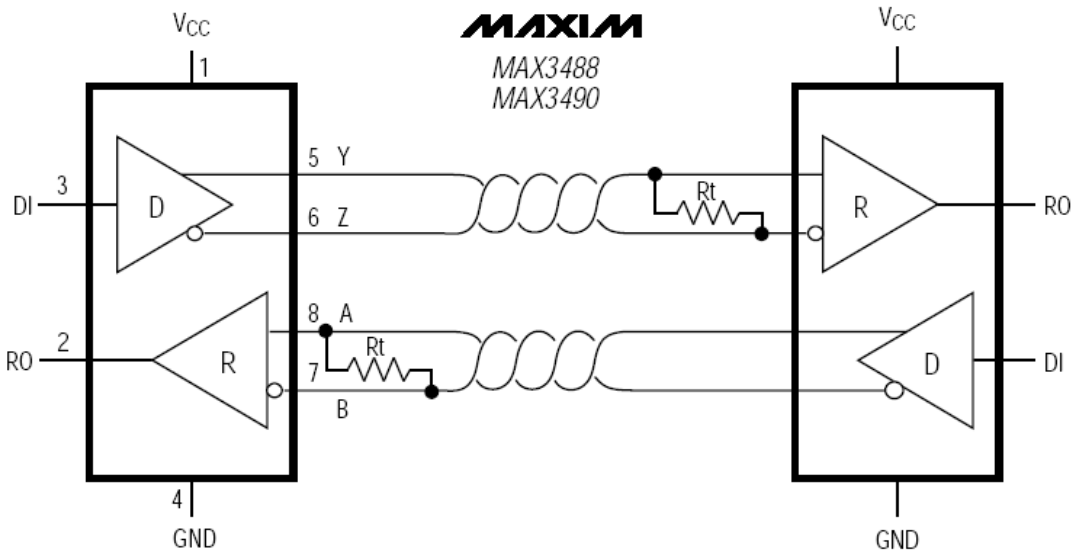


Why is the level shifting not done in the UART?

Why?

- UART manages data formatting not wire voltages and signals.
  - UART = Layer 2, formatting bits
  - Level Shifter = Layer 1, signaling
- The UART can be used with other media and line voltages
  - like RS-422 (twisted pair)
  - and IrDA.

For example: RS-422



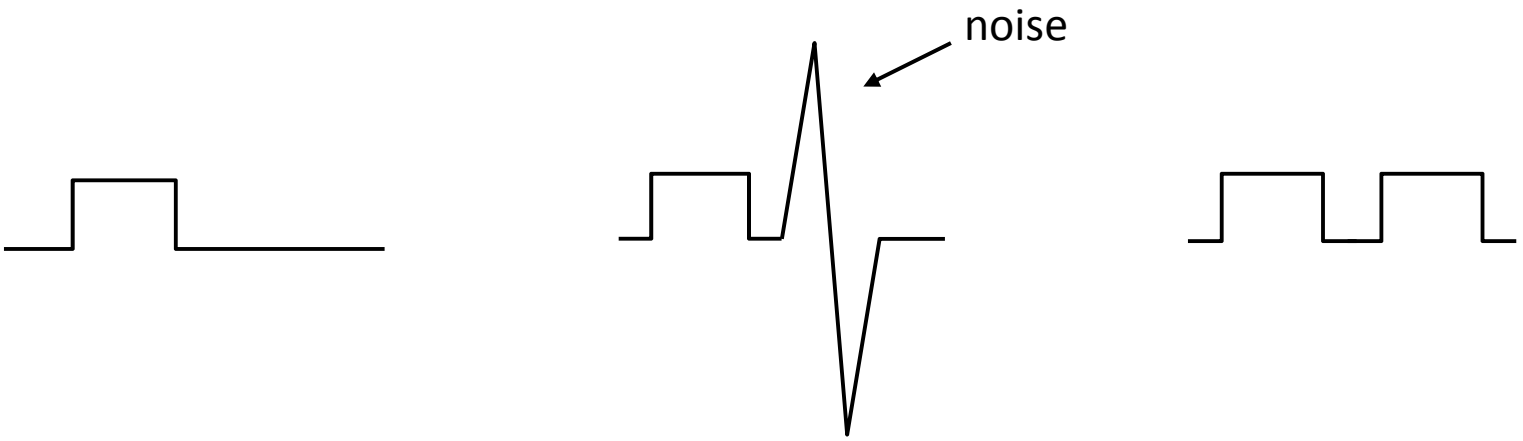
- RS 232 signals are represented by voltage levels with respect to ground.
- With RS 422 each signal uses a twisted pair for '**Differential voltage transmission**'. The signal is inactive when the voltage at A is negative and the voltage at B is positive.

What is differential signaling and why do we need it?

Differential Signaling

- Low-Voltage Differential Signaling (LVDS)
- A method of transmitting information over pairs of wires (as opposed to single-ended signaling, which uses a single wire).
- Reduces the noise on a connection by rejecting common-mode interference. Two wires are routed in parallel (sometimes twisted) so that they will receive the same interference. One wire carries the signal, and the other wire carries the inverse of the signal, so that the sum of the voltages on the two wires is always constant.

Single Ended Signaling

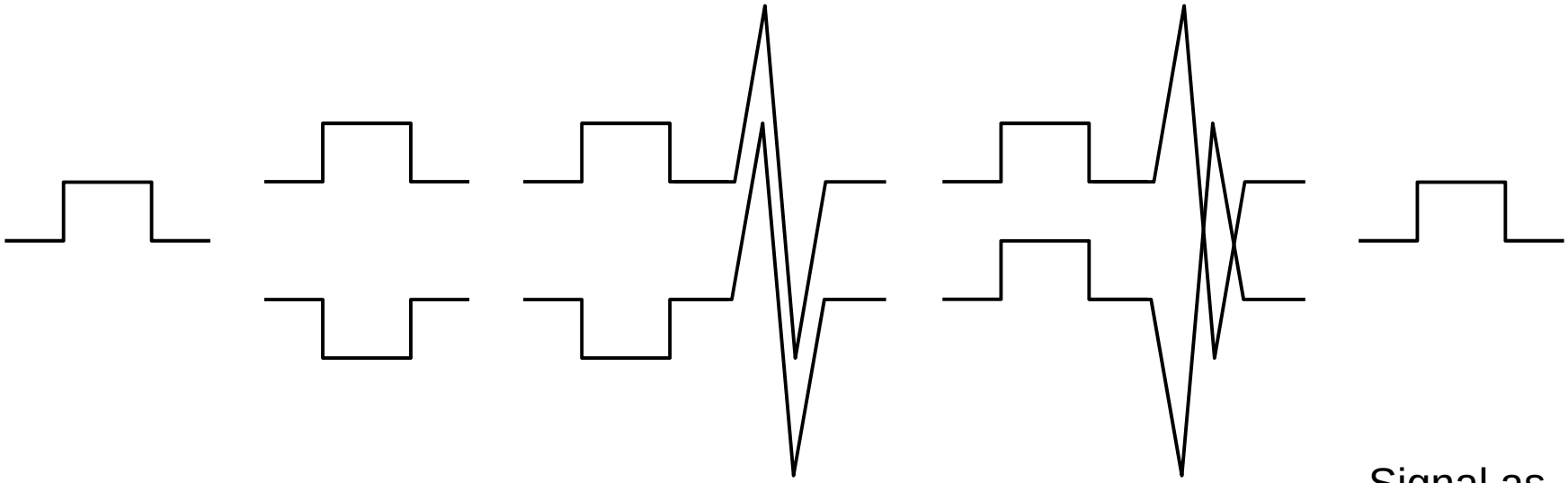


Signal as Transmitted

Signal on the wire with noise

Signal as received

Differential Signaling



Signal as Transmitted

Signal on the wire

Signal on the wire with noise

Receiver adds signal to inverse of signal. Noise is canceled.

Signal as received

Differential Signaling

- USB
- RS-422, RS-485
- Ethernet (on Cat 5)
- Controller Area Network (used in industrial and automotive applications)

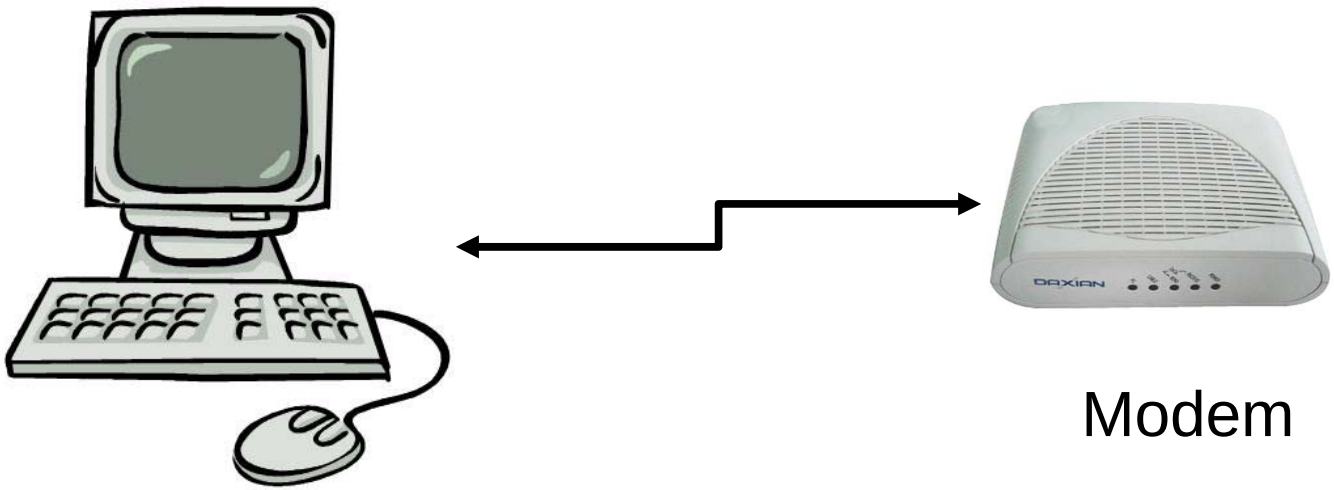
Serial Things

- Modems
- Bluetooth radios
- GPS Receivers
- Electronic Test Equipment (meters)
- Cell Phones

… lots of stuff

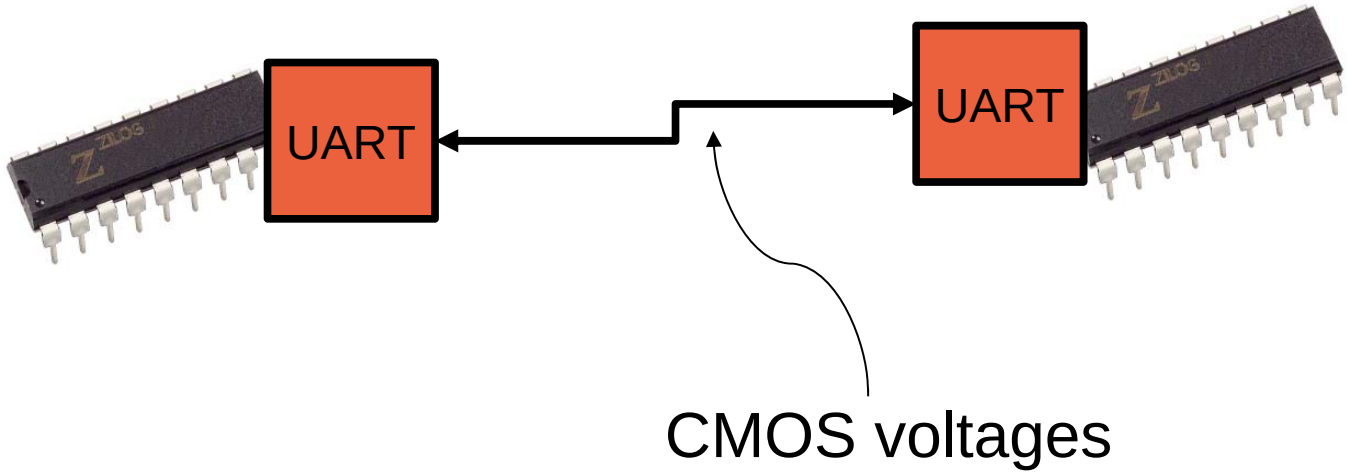
RS-232

- All consumer grade equipment has level shifters built-in.



Embedded Systems

- Embedded systems don't need to use level shifters at all (provided the serial connection is short enough, like a few inches of wire).



ZNEO UART

- 2 UARTS in the Z16 (U0 and U1)
- Fully configurable (ie: lots of registers)
  - 8-bit asynchronous data transfer
  - Selectable even- and odd-parity generation and checking
  - Option of one or two Stop bits
  - Separate transmit and receive interrupts
  - Framing, parity, overrun and break detection
  - Separate transmit and receive enables
  - Selectable 9-bit multiprocessor (9-bit) mode
  - 16-bit Baud Rate Generator (BRG)

LIN-UART

- Called the LIN-UART (Local Interconnect Network UART)
- Supports LIN protocol.

What is LIN?

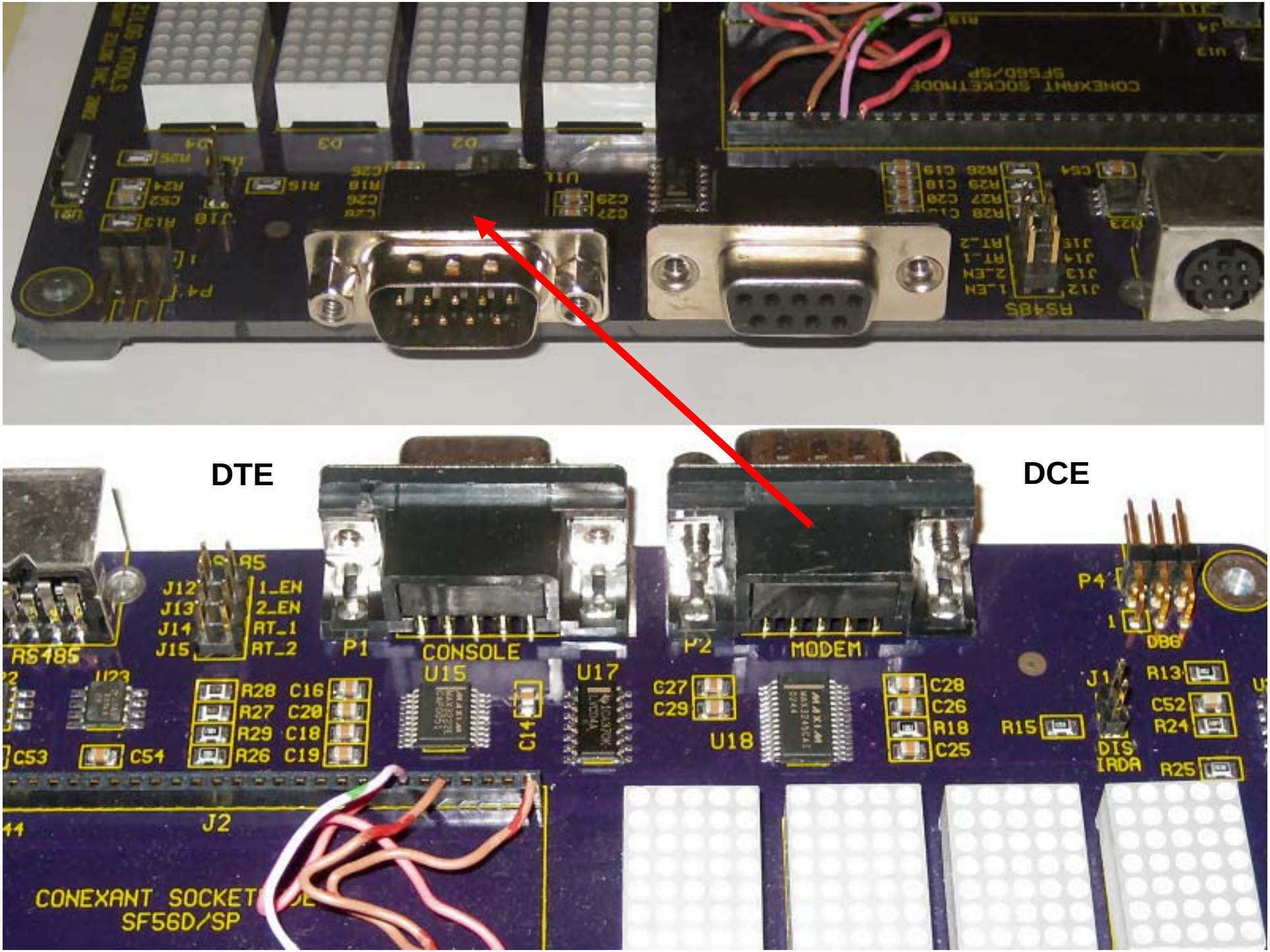
LIN

- LIN (Local Interconnect Network)
- LIN bus is an low cost serial protocol
- LIN is a broadcast serial network comprising one master and many (up to 16) slaves. No collision detection exists, therefore all messages are initiated by the master with at most one slave replying for a given message identifier.

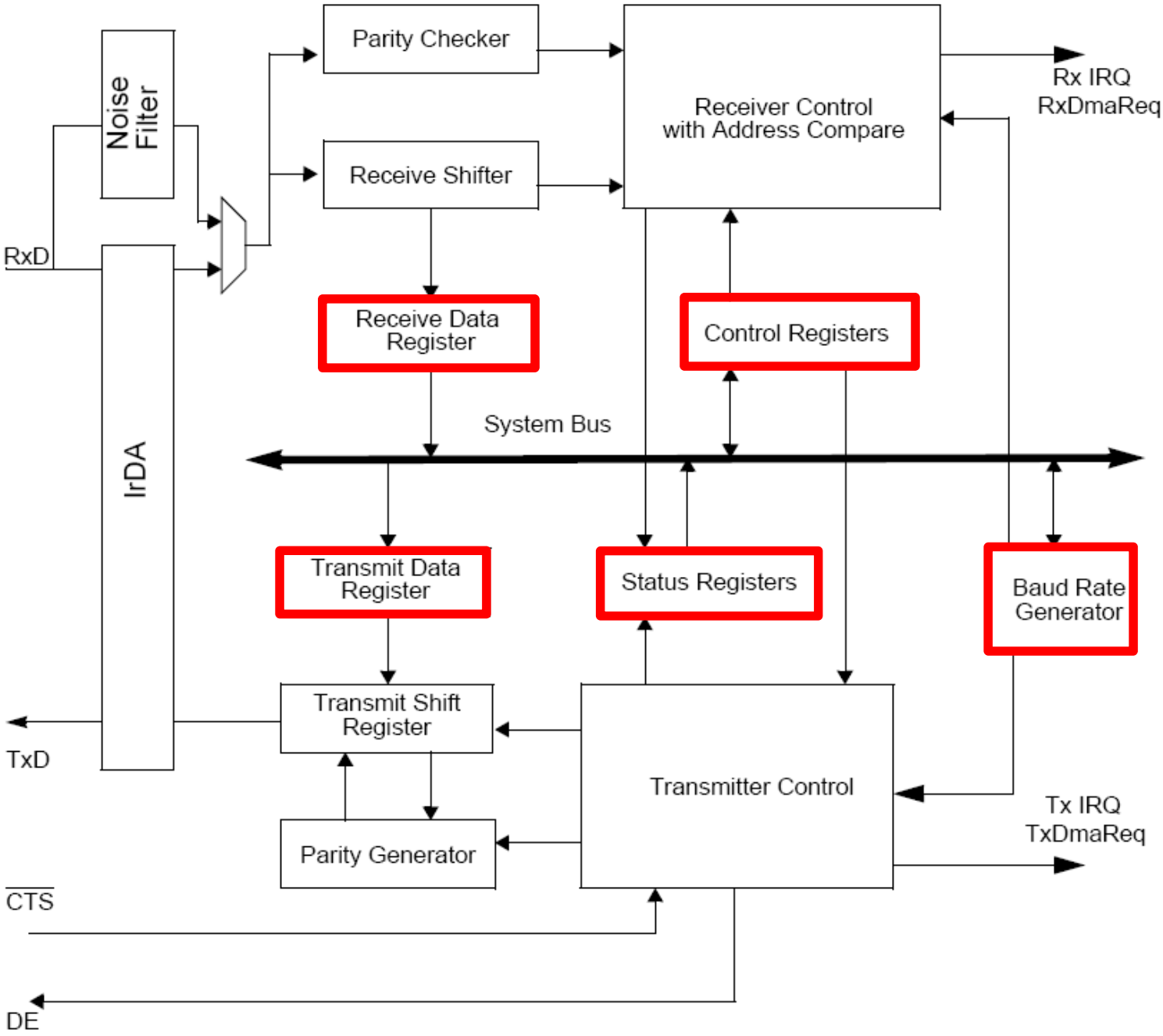
LIN

- It is particularly intended for mechatronic nodes in distributed automotive applications, but is also suited to industrial applications.
- It is intended to complement the existing CAN network leading to hierarchical networks within cars.

The ZNEO kit as 2 serial ports. Whats the difference between them?


DTE
DCE
P1
CONSOLE
P2
MODEM
U15
U17
U18
J12
J13
J14
J15
1_EN
2_EN
RT_1
RT_2
RS485
P4
DBG
J1
DIS
IRDA
R15
R25
J2
C53
C54
C14
C27
C29
C28
C26
R18
C25
R13
C52
R24
R28
C16
R27
C20
R29
C18
R26
C19
CONEXANT SOCKET
SF56D/SP

ZNEO UART

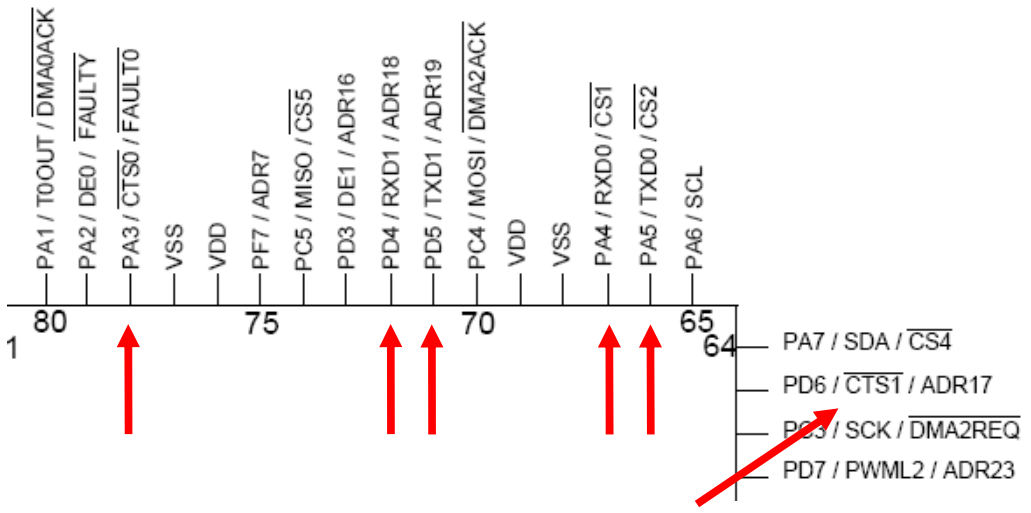

Parity Checker
Noise Filter
Receiver Control with Address Compare
Rx IRQ
RxDmaReq
Receive Shifter
RxD
Receive Data Register
Control Registers
System Bus
IrDA
Transmit Data Register
Status Registers
Baud Rate Generator
Transmit Shift Register
TxD
Transmitter Control
Tx IRQ
TxDmaReq
Parity Generator
CTS
DE

UART modes

- Transmitting Data using Polled Method
- Transmitting Data, Interrupt Driven Method
- Receiving Data using Polled Method
- Receiving Data, Interrupt Driven Method
- Receiving Data using the Direct Memory Access Controller (DMA)
- Multiprocessor Mode
- LIN mode

UART Pins

- PA3 - /CTS0
- PA4 - RXD0
- PA5 - TXD0
- PD6 - /CTS1
- PD4 – RXD1
- PD5 - TXD1



What about the other serial port control lines?

- Like DTR, DSR, CDC, RI …

UART Registers

- U0TXD – Data to be transmitted
- U0RXD – Data received
- U0STAT0 – UART status
- U0CTL0, U0CTL1 – UART control
- U0BRH, U0BRL – Baud Rate
- U0MDSTAT – Mode Select and Status
- U0ADDR – LIN address
- U1*

UxTXD, UxRXD

Table 80. LIN-UART Transmit Data Register (UxTXD)

BITS	7	6	5	4	3	2	1	0
FIELD	TXD							
RESET	X							
R/W	W							
ADDR	FF-E200H, FF-E210H							

TXD—Transmit Data

LIN-UART transmitter data byte to be shifted out through the TXD pin.

Table 81. LIN-UART Receive Data Register (UxRXD)

BITS	7	6	5	4	3	2	1	0
FIELD	RXD							
RESET	X							
R/W	R							
ADDR	FF-E200H, FF-E210H							

RXD—Receive Data

LIN-UART receiver data byte from the RXD pin

UxCTL0

Table 85. LIN-UART Control 0 Register (UxCTL0)

BITS	7	6	5	4	3	2	1	0
FIELD	TEN	REN	CTSE	PEN	PSEL	SBRK	STOP	LBEN
RESET	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADDR	FF-E202H, FF-E212H							

TEN—Transmit Enable
This bit enables or disables the transmitter. The enable is also controlled by the $\overline{\text{CTS}}$ signal and the CTSE bit. If the $\overline{\text{CTS}}$ signal is Low and the CTSE bit is 1, the transmitter is enabled.
0 = Transmitter disabled.
1 = Transmitter enabled.

REN—Receive Enable
This bit enables or disables the receiver.
0 = Receiver disabled.
1 = Receiver enabled.

CTSE—CTS Enable
0 = The $\overline{\text{CTS}}$ signal has no effect on the transmitter.
1 = The LIN-UART recognizes the $\overline{\text{CTS}}$ signal as an enable control for the transmitter.

PEN—Parity Enable

This bit enables or disables parity. Even or odd is determined by the PSEL bit.

0 = Parity is disabled. This bit is overridden by the MPEN bit.

1 = The transmitter sends data with an additional parity bit and the receiver receives an additional parity bit.

PSEL—Parity Select

0 = Even parity is transmitted and expected on all received data.

1 = Odd parity is transmitted and expected on all received data.

SBRK—Send Break

This bit pauses or breaks data transmission. Sending a break interrupts any transmission in progress, so ensure that the transmitter has finished sending data before setting this bit. In standard UART mode, the duration of the break is determined by how long software leaves this bit asserted. Also the duration of any required `Stop` bits following the break must be timed by software before writing a new byte to be transmitted to the transmit data register. In LIN mode, the master sends a Break character by asserting `SBRK`. The duration of the break is timed by hardware, and the SBRK bit is deasserted by hardware when the Break is completed. The duration of the Break is determined by the TxBreakLength field of the LIN control register. One or two Stop bits are automatically provided by the hardware in LIN mode as defined by the Stop bit.

0 = No break is sent.

1 = The output of the transmitter is 0.

STOP—Stop Bit Select

0 = The transmitter sends one stop bit.

1 = The transmitter sends two stop bits.

LBEN—Loop Back Enable

0 = Normal operation.

1 = All transmitted data is looped back to the receiver within the IrDA module.

UxSTAT0

Table 82. LIN-UART Status 0 Register – Standard UART Mode (UxSTAT0)

BITS	7	6	5	4	3	2	1	0
FIELD	RDA	PE	OE	FE	BRKD	TDRE	TXE	CTS
RESET	0	0	0	0	0	1	1	X
R/W	R	R	R	R	R	R	R	R
ADDR	FF-E201H, FF-E211H							

RDA—Receive Data Available

This bit indicates that the LIN-UART receive data register has received data. Reading the LIN-UART receive data register clears this bit.

0 = The LIN-UART receive data register is empty.

1 = There is a byte in the LIN-UART receive data register.

PE—Parity Error

This bit indicates that a parity error has occurred. Reading the receive data register clears this bit.

0 = No parity error occurred.

1 = A parity error occurred.

OE—Overrun Error

This bit indicates that an overrun error has occurred. An overrun occurs when new data is received and the receive data register has not been read. Reading the receive data register clears this bit.

0 = No overrun error occurred.

1 = An overrun error occurred.

FE—Framing Error

This bit indicates that a framing error (no Stop bit following data reception) is detected. Reading the receive data register clears this bit.

0 = No framing error occurred.

1 = A framing error occurred.

BRKD—Break Detect

This bit indicates that a break has occurred. If the data bits, parity/multiprocessor bit, and Stop bit(s) are all zeros then this bit is set to 1. Reading the receive data register clears this bit.

0 = No break occurred.

1 = A break occurred.

TDRE—Transmitter Data Register Empty

This bit indicates that the transmit data register is empty and ready for additional data. Writing to the transmit data register resets this bit.

0 = Do not write to the transmit data register.

1 = The transmit data register is ready to receive an additional byte to be transmitted.

TXE—Transmitter Empty

This bit indicates that the transmit shift register is empty and character transmission is finished.

0 = Data is currently transmitting.

1 = Transmission is complete.

CTS—$\overline{\text{CTS}}$ signal

When this bit is read it returns the $\overline{\text{CTS}}$ signal level. If `LBEN` = `1`, the $\overline{\text{CTS}}$ input signal is replaced by the internal receive data Available signal to provide flow control in loopback mode. CTS only affects transmission if the `CTSE` bit = 1.

UxBRH/L

Table 90. LIN-UART Baud Rate High Byte Register (UxBRH)

BITS	7	6	5	4	3	2	1	0
FIELD	BRH							
RESET	1							
R/W	R/W							
ADDR	FF-E206H, FF-E216H							

Table 91. LIN-UART Baud Rate Low Byte Register (UxBRL)

BITS	7	6	5	4	3	2	1	0
FIELD	BRL							
RESET	1							
R/W	R/W							
ADDR	FF-E207H, FF-E217H							

Note:

The UART must be disabled when updating the baud rate registers because high and low registers must be written independently.

UART Baud Rate

The LIN-UART data rate is calculated using the following equation for standard UART operation:

$$\textbf{UART Baud Rate (bits/s)} = \frac{\textbf{System Clock Frequency (Hz)}}{\textbf{16} \times \textbf{UART Baud Rate Divisor Value}}$$

For a given LIN-UART data rate, the integer baud rate divisor value is calculated using the following equation for standard UART operation:

$$\textbf{UART Baud Rate Divisor Value (BRG)} = \mathit{Round}\left(\frac{\textbf{System Clock Frequency (Hz)}}{\textbf{16} \times \textbf{UART Data Rate (bits/s)}}\right)$$

The baud rate error relative to the appropriate baud rate is calculated using the following equation:

$$\textbf{UART Baud Rate Error (\%)} = \textbf{100} \times \left(\frac{\textbf{Actual Data Rate} - \textbf{Desired Data Rate}}{\textbf{Desired Data Rate}}\right)$$

UART Interrupts

- The UART can generate the following interrupts:
  - UART0 receive
  - UART0 transmit
  - UART1 receive
  - UART1 transmit

Why would you want to get an interrupt on transmit?

Recall that the clock is 18.432 MHz

- Why that odd frequency, not something simple like 18 MHz?

20.0 MHz System Clock			
Desired Rate	BRG Divisor	Actual Rate	Error
(kHz)	(Decimal)	(kHz)	(%)
1250.0	1	1250.0	0.00
625.0	2	625.0	0.00
250.0	5	250.0	0.00
115.2	11	113.64	-1.19
57.6	22	56.82	-1.36
38.4	33	37.88	-1.36
19.2	65	19.23	0.16
9.60	130	9.62	0.16
4.80	260	4.81	0.16
2.40	521	2.399	-0.03
1.20	1042	1.199	-0.03
0.60	2083	0.60	0.02
0.30	4167	0.299	-0.01

1.8432 MHz System Clock			
Desired Rate	BRG Divisor	Actual Rate	Error
(kHz)	(Decimal)	(kHz)	(%)
1250.0	N/A	N/A	N/A
625.0	N/A	N/A	N/A
250.0	N/A	N/A	N/A
115.2	1	115.2	0.00
57.6	2	57.6	0.00
38.4	3	38.4	0.00
19.2	6	19.2	0.00
9.60	12	9.60	0.00
4.80	24	4.80	0.00
2.40	48	2.40	0.00
1.20	96	1.20	0.00
0.60	192	0.60	0.00
0.30	384	0.30	0.00

18.432 MHz →

1.8432 MHz System Clock			
Desired Rate	**BRG Divisor**	**Actual Rate**	**Error**
(kHz)	**(Decimal)**	**(kHz)**	**(%)**
1250.0	N/A	N/A	N/A
625.0	N/A	N/A	N/A
250.0	N/A	N/A	N/A
115.2	10	115.2	0.00
57.6	20	57.6	0.00
38.4	30	38.4	0.00
19.2	60	19.2	0.00
9.60	120	9.60	0.00
4.80	240	4.80	0.00
2.40	480	2.40	0.00
1.20	960	1.20	0.00
0.60	1920	0.60	0.00
0.30	3840	0.30	0.00

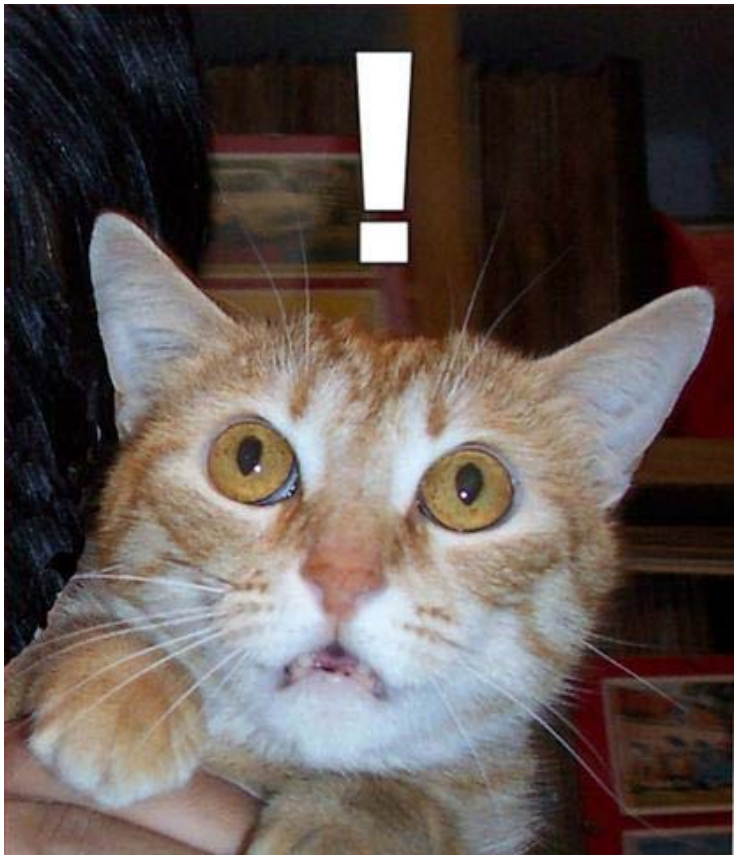
Don't forget ...

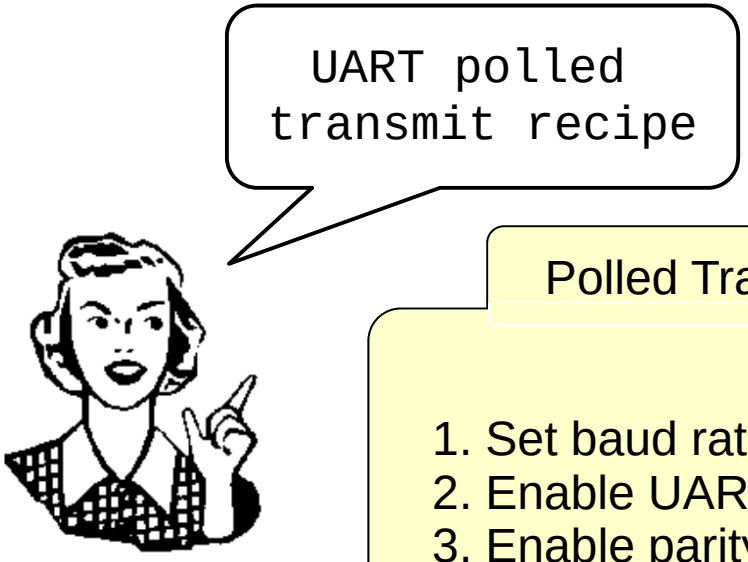
- Don't forget to enable the Alternate Function on port A or D for UAR0 or UART1, or there will be no data

Port	Pin	Alternate Function 1
Port A	PA0	T0IN / $\overline{\text{T0OUT}}$
	PA1	T0OUT
	PA2	DE0
	PA3	$\overline{\text{CTS0}}$
	PA4	RXD0
	PA5	TXD0
	PA6	SCL
	PA7	SDA

Port	Pin	Alternate Function 1
Port D	PD0	PWMH1
	PD1	PWML1
	PD2	PWMH2
	PD3	DE1
	PD4	RXD1
	PD5	TXD1
	PD6	$\overline{\text{CTS1}}$
	PD7	PWML2

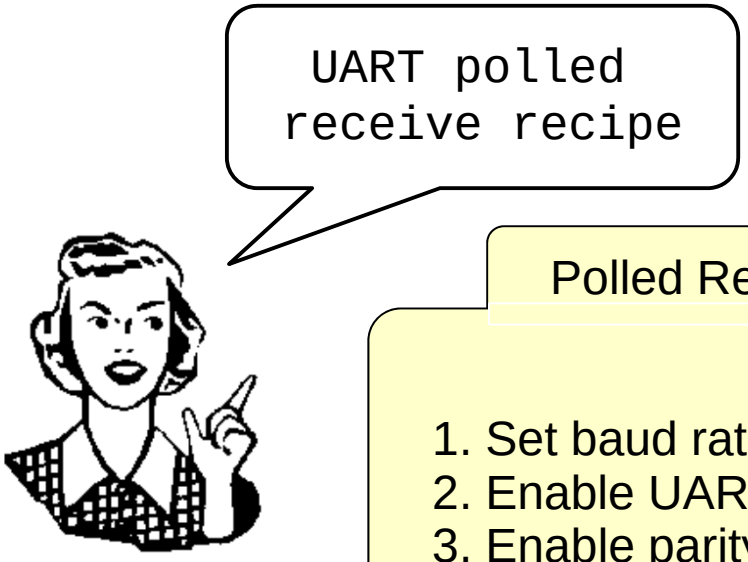
No Multiprocessor or LIN mode discussion!





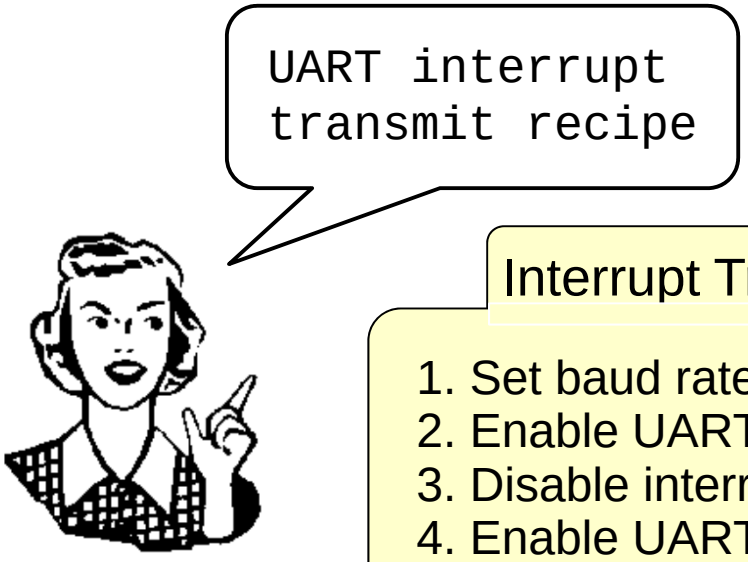
Polled Transmit

1. Set baud rate (UxBRH/L)
2. Enable UART pin functions (UxCTL0,1)
3. Enable parity if desired (PEN and PSEL bits in UxCTL0)
4. Enable UART by setting TEN bit in UxCTL0
5. Set alternate function on GPIO pins
6. Set or clear CTSE bit in UxCTL0 to enable flow control on the /CTS pin
7. When TDRE bit in UxSTAT0 is set (1) then the transmit buffer is empty, write one byte to UxTXD to send byte. Repeat this step until all bytes have been sent.



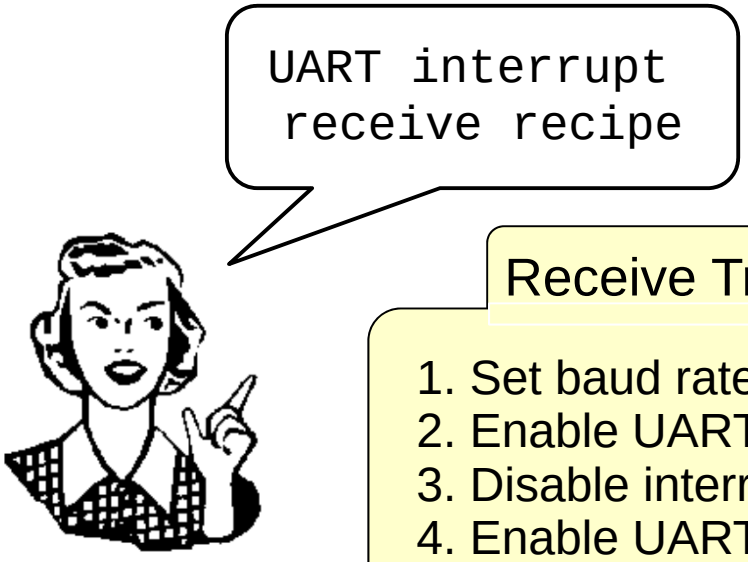
Polled Receive

1. Set baud rate (UxBRH/L)
2. Enable UART pin functions (UxCTL0/1)
3. Enable parity if desired (PEN and PSEL bits in UxCTL0)
4. Set alternate function on GPIO pins
5. Enable UART by setting REN bit in UxCTL0
6. When RDA bit in UxSTAT0 is set (1) then the transmit buffer is full, read one byte from UxRXD to get byte. Repeat this step until all bytes have been received.



Interrupt Transmit

1. Set baud rate (UxBRH/L)
2. Enable UART pin functions (UxCTL0/1)
3. Disable interrupts
4. Enable UARTx transmitter interrupts by writing to the Interrupt control registers.
5. Enable UART by setting TEN bit in UxCTL0
6. Set alternate function on GPIO pins
7. Enable parity if desired (PEN and PSEL bits in UxCTL0)
8. Set or clear CTSE bit in UxCTL0 to enable flow control on the /CTS pin
9. Enable interrupts
10. Write your ISR
11. ISR should write byte to UxTXD and then clear the UART transmit bit in the Interrupt Control register
12. Assign the ISR vector.

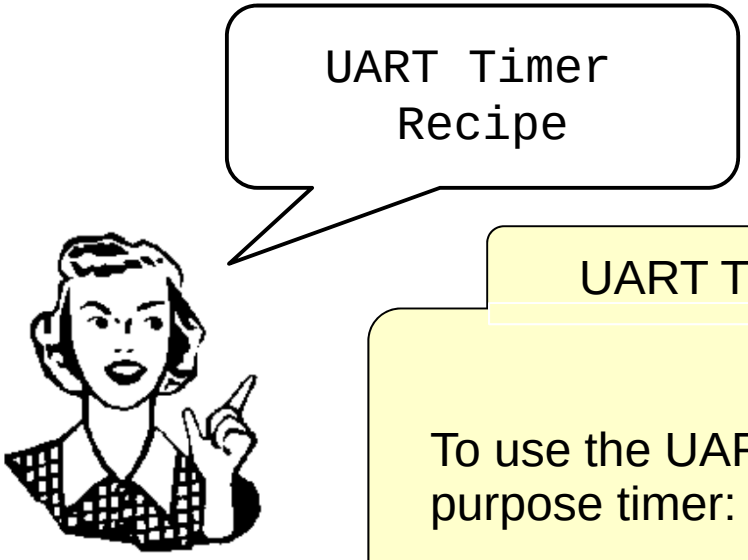


Receive Transmit

1. Set baud rate (UxBRH/L)
2. Enable UART pin functions (UxCTL0/1)
3. Disable interrupts
4. Enable UARTx receiver interrupts by writing to the Interrupt control registers.
5. Enable UART by setting REN bit in UxCTL0
6. Set alternate function on GPIO pins
7. Enable parity if desired (PEN and PSEL bits in UxCTL0)
8. Enable interrupts
9. Write your ISR
10. ISR should check the UxSTAT0 byte to see if there are any errors. If not, write byte to UxRXD and then clear the UART receive bit in the Interrupt Control register
11. Assign the ISR vector.

An extra timer?

- Use the UART baud rate generator as a timer (when you need more timers and don't need to use the serial port).



UART Timer

To use the UART Baud Rate Generator as a general purpose timer:

1. Disable UART (clear the REN and TEN bits in UxCTL0)
2. Compute the 16 bit baud rate value

$$\text{UART Baud Rate Divisor Value (BRG)} = \text{Round}\left(\frac{\text{System Clock Frequency (Hz)}}{16 \times \text{UART Data Rate (bits/s)}}\right)$$

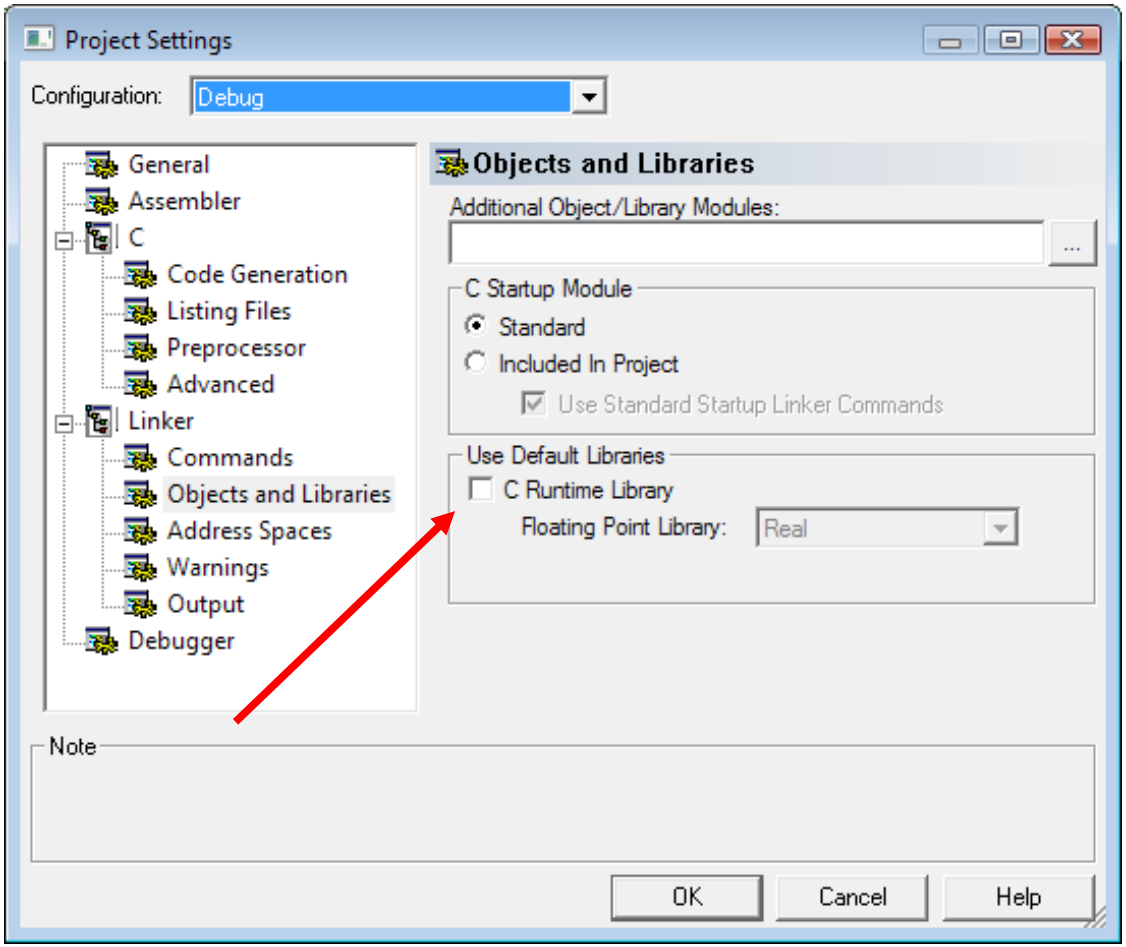
3. Load UxBRH and UxBRL
4. Write your ISR
5. Set the interrupt vector (UARTx receive)
6. Enable the baud rate generator timer and interrupt by setting BIRQ bit in UxCTL1
7. Do NOT set alternate function on GPIO pins!
8. Enjoy the extra timer

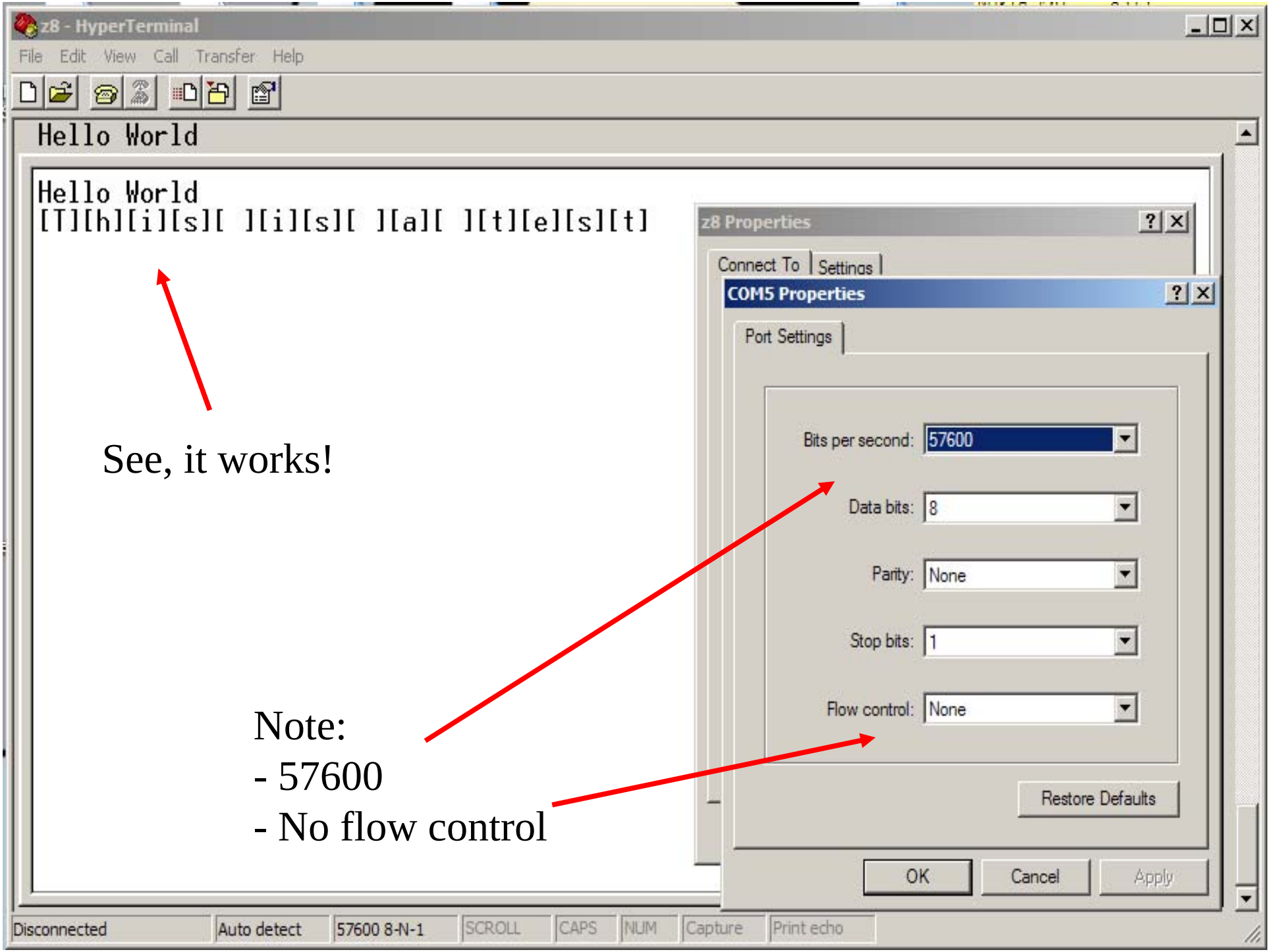
First Example

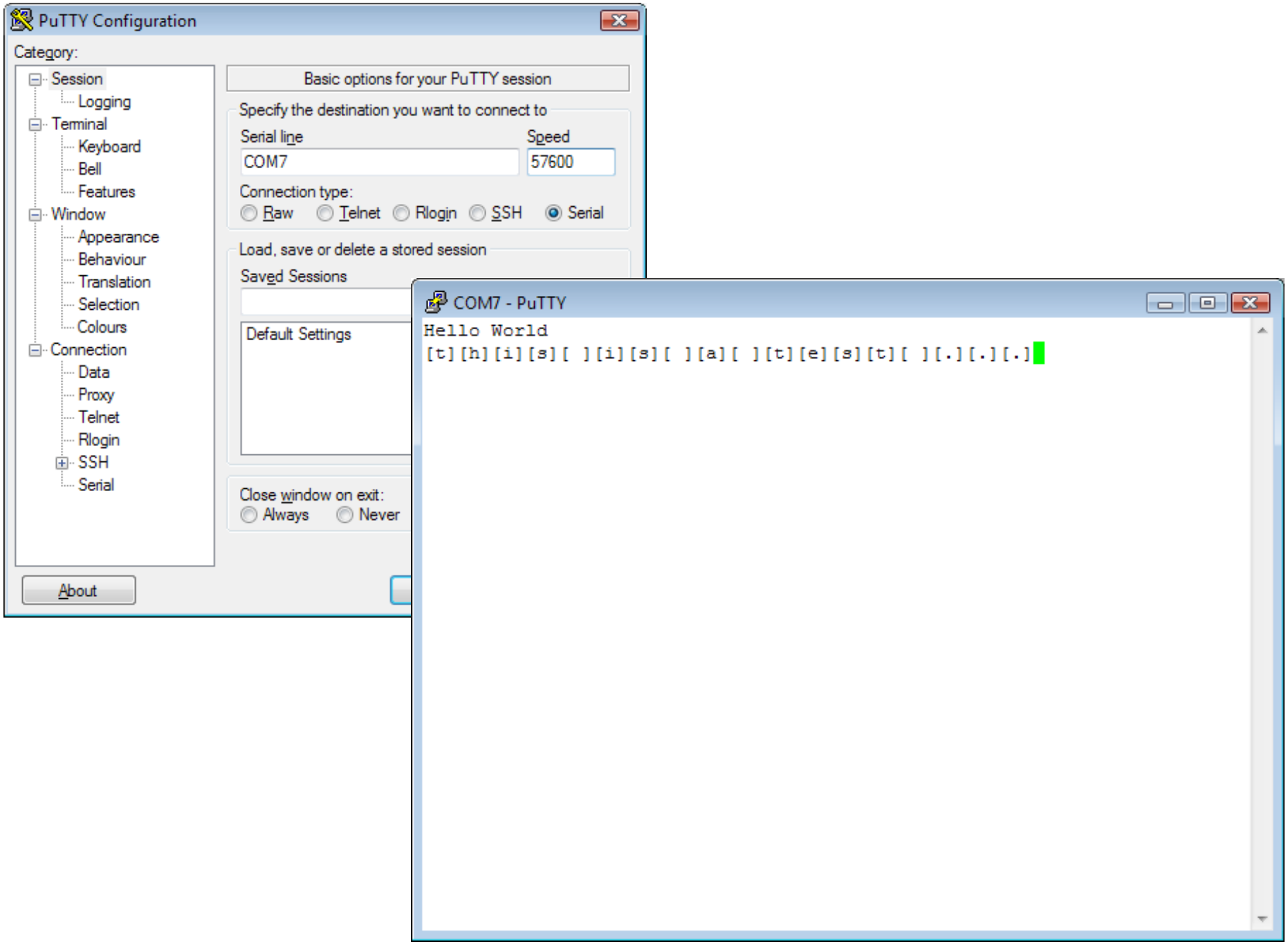
Example_
SerialHard

- Demonstrate how to send and receive characters by controlling the UART

No C Runtime needed!




z8 - HyperTerminal
File Edit View Call Transfer Help
Hello World
Hello World
[T][h][i][s][ ][i][s][ ][a][ ][t][e][s][t]
See, it works!
Note:
- 57600
- No flow control
z8 Properties
Connect To
Settings
COM5 Properties
Port Settings
Bits per second: 57600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None
Restore Defaults
OK
Cancel
Apply
Disconnected
Auto detect
57600 8-N-1
SCROLL
CAPS
NUM
Capture
Print echo


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Serial line
Speed
COM7
57600
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
Close window on exit:
Always
Never
About
COM7 - PuTTY
Hello World
[t][h][i][s][ ][i][s][ ][a][ ][t][e][s][t][ ][.][.][.]

UART functions

- Fortunately <stdio.h> and <sio.h> help with all/most of this.

sio.h

Nonstandard I/O Functions <sio.h>

This header contains nonstandard ZNEO-specific input/output functions.

_DEFFREQ	Expands to unsigned long default frequency.
_DEFBAUD	Expands to unsigned long default baud rate.
_UART0	Expands to an integer indicating UART0.
_UART1	Expands to an integer indicating UART1.

Functions

char getch(void) ;	Returns the data byte available in the selected UART.
int init_uart(int port,unsigned long freq, unsigned long baud);	Initializes the selected UART for specified settings and returns the error status.
int kbhit(void);	Checks for receive data available on selected UART.
int putch(char) ;	Sends a character to the selected UART and returns the error status.
int select_port(int port);	Selects the UART. Default is _UART0.

stdio.h

Input/Output <stdio.h>

The <stdio.h> header declares input and output functions.

Macro

EOF	Expands to a negative integral constant. Returned by functions to indicate end of file.

Functions

Formatted Input/Output

int printf(const char *format, ...);	Writes formatted data to stdout.
int scanf(const char *format, ...);	Reads formatted data from stdin.
int sprintf(char *s, const char *format, ...);	Writes formatted data to string.
int sscanf(const char *s, const char *format, ...);	Reads formatted data from string.
int vprintf(const char *format, va_list arg);	Writes formatted data to a stdout.
int vsprintf(char *s, const char *format, va_list arg);	Writes formatted data to a string.

stdio.h

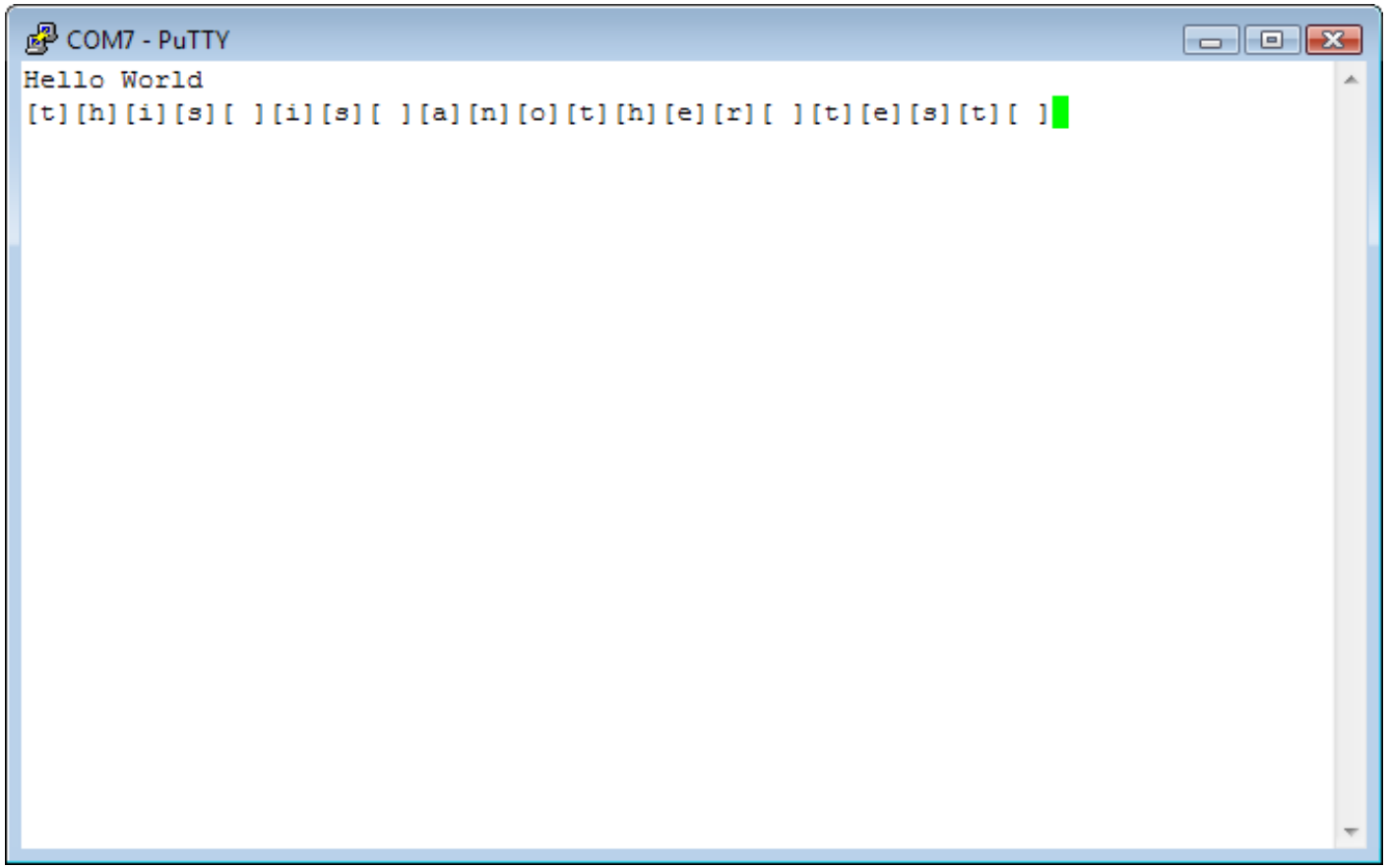
Character Input/Output

int getchar(void);	Reads a character from stdin.
char *gets(char *s);	Reads a line from stdin.
int putchar(int c);	Writes a character to stdout.
int puts(const char *s);	Writes a line to stdout.

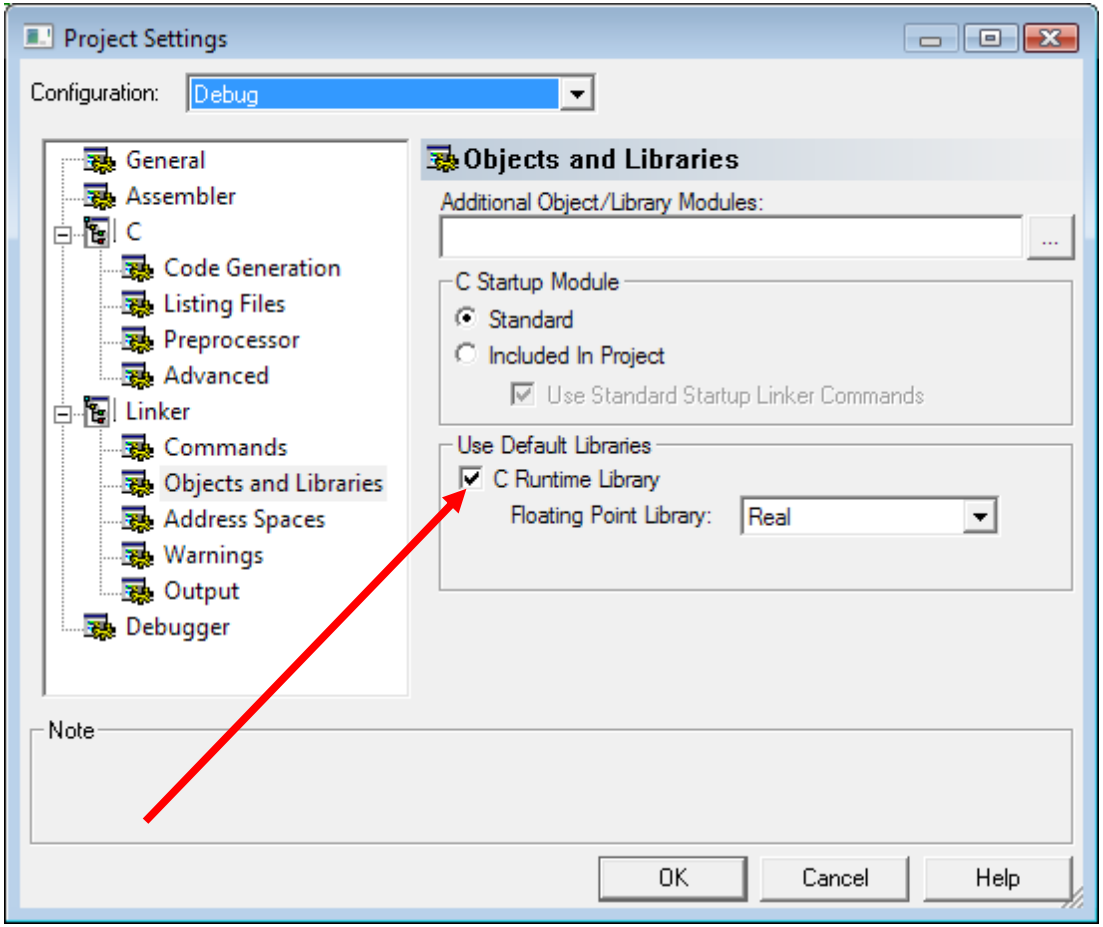
Another Example

Example_
SerialSimple

- Demonstrate how to send and receive characters by using sio.h and stdio.h (who control the UART for us)


COM7 - PuTTY
Hello World
[t][h][i][s][ ][i][s][ ][a][n][o][t][h][e][r][ ][t][e][s][t][ ]

This Time ...


Project Settings
Configuration:
Debug
General
Assembler
C
Code Generation
Listing Files
Preprocessor
Advanced
Linker
Commands
Objects and Libraries
Address Spaces
Warnings
Output
Debugger
Objects and Libraries
Additional Object/Library Modules:
C Startup Module
Standard
Included In Project
Use Standard Startup Linker Commands
Use Default Libraries
C Runtime Library
Floating Point Library:
Real
Note
OK
Cancel
Help

If not

- If the C runtime library is not included you will see this less than helpful error message:

```
Linking...
FATAL ERROR (724) --> Symbol
  _getchar (.\main_easy.obj) is
  not defined.
Build completed.
```

Last Word (on serial ports)

- The Serial Port overview, was the "light-n-fluffy" version.
- For much more detailed information:
  - http://www.beyondlogic.org/serial/serial.htm
  - http://www.tldp.org/HOWTO/Serial-HOWTO.html
  - http://www.lvr.com/serport.htm
  - ...

Why might you (the student) want to implement a serial receive function that uses interrupts on receipt of data?